Is it a bird? Is it a plane?

DiPS CodeJam 24

Prompt

You're a researcher deep in the jungles of the Amazon plotting flight paths for a few rare birds. But there's a problem – planes! Planes seem to be interfering with your plotting of flight paths, and you need to filter them out to get your data. Every flight path that you capture, plane or bird, is denoted by a space-separated string of coordinates for each stop the object makes in the format $T(x, y)$, where T denotes the type of stop (short, medium, long) and x and y denote the coordinates of the stop. Birds follow a pattern in their stops, either in terms of cycling between the coordinates or cycling through a pattern of durations of their stops, while planes do not follow any discernible pattern. For example:
$S0, 0$ $L0, 2$ $S0, 3$ $L0, 4$ is a bird,
$S0, 0$ $L0, 4$ $M0, 3$ $M0, 0$ $M0, 4$ $S0, 3$ is a bird,
$S0, 0$ $L0, 1$ $S0, 3$ $M0, 0$ $L0, 4$ $S0, 2$ $M0, 5$ is a plane

Can you find how many birds you've captured?

Input Format

- The first line of the input contains an integer n, denoting the number of flight paths.
- The next n lines of input each contain a flight path.

Output Format

The first and only line of your output must contain a single integer m, denoting the number of birds.

Constraints

- $100 \leq n \leq 1000$
- $x, y \leq 10^3$
- The number of coordinates in each path is between 100 and 1000

Solution

Sample Program

```
def factors(n):
  return list(set(reduce(list.__add__,
                ([i, n//i] for i in range(1, int(n**0.5) + 1) if n % i == 0))))

def solve(s):
  path = [[i.split(",")[0][0], int(i.split(",")[0][1::]), int(i.split(",")[1])] for i
    in s.split()]
```

```
durations = [i[0] for i in path]
coords = [(i[1], i[2]) for i in path]

def can_construct_from(sub, s):
  return sub * (len(s) // len(sub)) == s

# check for patterns in duration:
for L in range(1, len(durations) // 2 + 1):
  if len(durations) % L == 0:  # L must be a divisor of n
    sub = durations[:L]
    if can_construct_from(sub, durations):
      return True

# check for patterns in coordinates:
for L in range(1, len(coords) // 2 + 1):
  if len(coords) % L == 0:  # L must be a divisor of n
    sub = coords[:L]
    if can_construct_from(sub, coords):
      return True

return False
```